**Southern New Hampshire University**

**Kewanee McGhee Sept 11th, 2022**

**CS 320 Assignment: 7-2 Project Submission**

The unit testing approach, I implemented were examples of white-box testing and black-box testing. It was difficult at first, but almost 100% of the time whether through my rubric or doing something a little funky on my own. I had to reach out for help several times with writing the service classes and tests. I would pass in an assignment and think, “I got it!” Only to find out I may have been creating too much code, which would create further testing! From the rubric and requirements of the assignments, I did try to the best of my ability, having no prior knowledge to complete the tasks. One example was to code the Appointment class. The appointment ID could not be longer than ten characters, and it could not be null or updatable. I created private variables as well creating setters and getters, and then lastly a constructor that ties everything together. Well, too many constructors. I learned after my last assignment was graded. The setters, I implemented a condition statement that if the input is null, then it will show an error message or past a certain number of characters, if it wasn’t null then it should add the requirement. I applied for all the other classes too! The overall quality of the JUnit tests would be 100% with one in the 93% area. I hadn’t realized I was supposed to test for failed tests. I didn’t test that check the exceptions for invalid input.

I did learn so much! I did not have any issues with the testing once I figured out how to setup up the code. I could improve on is testing for a scenario that fails. In the future I plan to practice more testing. It is especially important to test for failed tests. In my assignment, had great tests for the validation methods, but the actual throwing of the exception is not tested. I did not need to validate that the phone was all digits. I was told I may have added too much, and “more you add, the more you need to test, and the more that can break” (Kalinowski, Neil)

It broke so much! The service should have taken in a first name, last name, phone, and address for an "add" and then create the Contact itself, choosing an ID. the service should be handling the IDs. I was instructed that an uncomplicated way to create an ID is to have an int counter as a class variable in the service that you increment on each "add" and then convert it to a string when passing it to the Contact constructor. My, -Add tests and -Add tests for invalid input for when the ID was not found when updating was not found. I decided to look up a few different examples on YouTube, then tried to produce my own version, but I do know practice makes perfect! I had a better time writing Junit tests and somehow, I got it to work.

In my AppointmentTest.java example:

class AppointmentTest {

final private byte APPOINTMENT\_ID\_LENGTH;

final private byte APPOINTMENT\_DESCRIPTION\_LENGTH;

final private String INITIALIZER;

private String appointmentId = "1234567890";

private Date appointmentDate = new Date();

private String description = "Doctor Appointment";

{

APPOINTMENT\_ID\_LENGTH = 10;

APPOINTMENT\_DESCRIPTION\_LENGTH = 50;

INITIALIZER = "INITIAL";

appointmentDate = getFutureDateFromToday(1);

}

@Test

void testAppointment() {

Appointment a = new Appointment();

assertEquals(INITIALIZER, a.getAppointmentId());

assertNotNull(a.getAppointmentDate());

assertEquals(INITIALIZER, a.getDescription());

}

@Test

void testAppointmentString() {

Appointment a = new Appointment(appointmentId);

assertEquals(appointmentId, a.getAppointmentId());

assertNotNull(a.getAppointmentDate());

assertEquals(INITIALIZER, a.getDescription());

}

@Test

void testAppointmentStringDate() {

Appointment a = new Appointment(appointmentId, appointmentDate);

assertEquals(appointmentId, a.getAppointmentId());

assertEquals(appointmentDate, a.getAppointmentDate());

assertEquals(INITIALIZER, a.getDescription());

}

@Test

void testAppointmentStringDateString() {

Appointment a = new Appointment(appointmentId, appointmentDate, description);

assertEquals(appointmentId, a.getAppointmentId());

assertEquals(appointmentDate, a.getAppointmentDate());

assertEquals(description, a.getDescription());

}

@Test

void testGetAppointmentId() {

}

@Test

void testUpdateDate() {

}

@Test

void testGetAppointmentDate() {

}

@Test

void testUpdateDescription() {

}

@Test

void testGetDescription() {

}

@Test

void test\_appointment\_ID() {

// check error message when id is null

Exception except\_null = assertThrows(IllegalArgumentException.class, () -> {

new Appointment(null);

});

assertEquals("Appointment ID cannot be null.", except\_null.getMessage());

// check error message when lenght of id is longer than ten

Exception except\_longer = assertThrows(IllegalArgumentException.class, () -> {

new Appointment("12345678901");

});

assertEquals("Appointment ID cannot exceed 10 characters.", except\_longer.getMessage());

}

@Test

void test\_appointment\_Date() {

// check error message when date is null

Exception except\_null = assertThrows(IllegalArgumentException.class, () -> {

new Appointment("123", null);

});

assertEquals("Appointment date cannot be null.", except\_null.getMessage());

// check error message when date is in the past

DateFormat formatter = new SimpleDateFormat("yyy-MM-dd");

Exception except\_past = assertThrows(IllegalArgumentException.class, () -> {

new Appointment("1234", formatter.parse("2022-05-16"));

});

assertEquals("Cannot make appointment in the past.", except\_past.getMessage());

}

@Test

void test\_appointment\_Description() {

// check error message when description is null

Appointment app = new Appointment();

Exception except\_null = assertThrows(IllegalArgumentException.class, () -> {

app.updateDescription(null);

});

assertEquals("Appointment description cannot be null.", except\_null.getMessage());

// check error message when length of description is longer than fifty

String invalidStr = "This is Invalid Description.This is Invalid Descrip";// length is 51

Exception except\_longer = assertThrows(IllegalArgumentException.class, () -> {

app.updateDescription(invalidStr);

});

assertEquals("Appointment description cannot exceed 50 characters.", except\_longer.getMessage());

}

private Date getFutureDateFromToday(int daysToAdd) {

Calendar cal = GregorianCalendar.getInstance();

cal.add(Calendar.DAY\_OF\_YEAR, daysToAdd);

return cal.getTime();

}

private Date getPastDate(int daysToDeduct) {

Calendar cal = GregorianCalendar.getInstance();

cal.add(Calendar.DAY\_OF\_YEAR, -1 \* daysToDeduct);

return cal.getTime();

}

}

This kind of project was a different experience for me and I had to discover a new way of thinking. Then I could properly run and apply it to the code that I developed. It was a new way of thinking but with each error, I was able to get a sense of and complete most of the testing. I was afraid to make mistakes but realized the experience in the mistakes is the reason for testing. I did not take into consideration various values for testing because I was so caught up in keeping up with the requirements. I did take into consideration whether the objects I created passed, but I didn’t check for if the failed, and coded too many numbers that didn’t need to be included within some of the services.

Reference:

Instructor: Neil Kalinowski, SNHU: [CS-320-T6627 Software Test Automation& QA 22EW6](https://learn.snhu.edu/d2l/home/1116026)